Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Andreja Kovačič

Detekcija prometnih znakov s konvolucijskimi nevronskimi mrežami

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Danijel Skočaj

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Detekcija prometnih znakov na slikah je zaradi svoje aplikativne uporabnosti pogosto obravnavan raziskovalni problem. V zadnjih letih se za razpoznavanje predmetov, vključno s prometnimi znaki, uporabljajo predvsem metode, ki temeljijo na konvolucijskih nevronskih mrežah. Na voljo so tudi metode, ki ne le razpoznajo vsebino regije slike, temveč tudi zaznajo potencialno zanimive regije na slikah. Ena najbolj uveljavljenih metod je metoda Faster R-CNN. Uporabite to metodo za reševanje problema detekcije prometnih znakov. Implementirajte nekatere razširitve in dodobra evalvirajte dobljen sistem. Preverite kako na rezultat vplivajo velikost in narava validacijske množice, velikost učne množice ter bolj napreden postopek izbora učnih primerov, ki temelji na ocenjevanju pomembnosti posameznih slik.

Hvala mentorju, Danijelu Skočaju, in Domnu Taberniku, za vso vodstvo in pomoč. Hvala tudi mojim najbližjim za vso podporo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CNN	convolutional neural network	konvolucijska nevronska mreža
RPN	region proposal network	generator regij
FC	fully connected	polno povezana
OHEM	online hard example mining	sprotno iskanje težkih primerov

Povzetek

Naslov: Detekcija prometnih znakov s konvolucijskimi nevronskimi mrežami

Avtor: Andreja Kovačič

Diplomsko delo se ukvarja z detekcijo in prepoznavanjem prometnih znakov z metodo Faster R-CNN. Razišče možnost uporabe generiranih podatkov za validacijo učenja, kot odgovor na omejeno velikost učne množice. Ker Faster R-CNN zaradi načina učenja ne dopušča običajnega učenja na težkih primerih (ang. bootstrapping), uporabimo novo metodo - sprotno iskanje težkih primerov. Na koncu iščemo optimalen način doučenja že obstoječega modela.

Ključne besede: Faster R-CNN, klasifikacija, detekcija, sprotno iskanje težkih primerov, doučenje,prometni znaki.

Abstract

Title: Traffic sign detection with convolutional neural networks

Author: Andreja Kovačič

The goal of this thesis is to describe and use the method Faster R-CNN for detection and recognition of traffic signs. It explores the possibility of using artificially generated images in validation set, in hopes of saving real images for train set. We tackle a real world problem of growing dataset through time. We'll try to find an optimal way to augment the already learned model with new images. Lastly, we try to apply a new method, online hard example mining, which is essentially bootstrapping for end-to-end systems.

Keywords: Faster R-CNN, classification, detection, online hard example mining, fine-tuning, traffic signs.

1. Uvod

V računalništvu se zadnja desetletja pospešeno ukvarjamo z možnostmi umetne inteligence. Začelo se je z obravnavanjem algebraičnih izrazov, ekspertnimi sistemi, igro šaha. Lani so časopise polnili članki o prvem porazu človeka v igri Go, ki velja za najbolj kompleksno namizno igro. A zdi se, da moramo za pravo inteligenco računalniku omogočiti opazovanje zunanjega sveta, skozi vizualno zaznavanje. Trenutno se zdijo za to naša najboljša možnost konvolucijske nevronske mreže, zato tudi poplava del, ki jih s pridom uporabljajo v najrazličnejše namene.

V tem delu jih bomo uporabili za detekcijo in prepoznavo prometnih znakov. Takšna aplikacija se lahko uporablja v vozilih v pomoč vozniku ali za avtonomno vožnjo, primer uporabe vidimo na sliki 1.1.

Konkretno, naša aplikacija bo služila za detekcijo znakov na slikah, ne nujno v realnem času. Hitrost zaznave tako ni zelo pomembna, pomembnejša je čim boljša zaznava znakov, ki se pojavljajo v različnih velikostih. Zaradi teh zahtev se zdi model Faster R-CNN primeren, saj je to trenutno ena izmed najuspešnejših metod za klasifikacijo objektov. Metodo bomo evalvirali na bazi slik prometnih znakov, ki vsebuje 160 kategorij različnih težavnosti. Radi bi se naučili prepoznavati povsem enostavne oziroma jasno določene znake, kot so STOP znak in prednostna cesta, kot tudi tiste, kjer je vsebina variabilna. Krajevne table, smerokazi in podobni znaki ne smejo zmesti našega sistema, oziroma je to nezaželeno, čeprav je izziv prepoznave takih znakov bralcu očiten. Poskušali bomo izboljšati učenje s sprotnim iskanjem težkih primerov, preizkusili pa bomo tudi nekaj metod za spopadanje

Slika 1.1: Prikaz pravilno klasificiranih znakov, označenih z zeleno.

s povsem realnimi problemi: majhno učno množico in povečanjem te skozi čas.

Veliko del je posvečenih prepoznavanju objektov s pomočjo konvolucijskih mrež [13, 7, 16, 2, 1, 16, 13]. Metoda Faster R-CNN sloni na detekciji objektov na regijah slik [10, 9, 3], uporablja tudi deljenje konvolucijskih plasti, ki so predmet večih raziskav, zaradi izboljšanja rezultatov detekcij [18, 7].

Tudi s problemom detekcije prometnih znakov smo se ljudje že spoprijemali [4, 20, 23].

V pričujočem delu bomo najprej podrobno predstavili Faster R-CNN, nato bomo predstavili teoretično plast eksperimentov, nazadnje bomo predstavili rezultate in izvedli analizo.

2. Faster R-CNN

2.1 Splošno o nevronskih mrežah

Nevronske mreže so poskus programerjev, da bi se z oponašanjem delovanja možganov približali resnični inteligenci. Resnici na ljubo smo vzeli precej poenostavljen model, vendar vseeno dosegajo dobre rezultate za širok nabor nalog. V grobem so možgani sestavljeni iz nevronov, ki so med seboj povezani. Po teh povezavah potujejo impulzi, ki se širijo samo če presežejo neko pragovno vrednost. Tudi pri umetnih nevronskih mrežah imamo nevrone, povezave med njimi in aktivacijske funkcije, ki vnašajo nelinearnost. Slika je v računalniku predstavljena kot matrika vrednosti slikovnih elementov (ang. pixels). Vhodni nevroni sprejmejo vsako od teh vrednosti, jih pošljejo naprej v drugo plast nevronov, kjer se njihova vrednost pomnoži z utežjo njihove povezave do nevrona. Vse vhodne vrednosti seštejemo in na njih apliciramo aktivacijsko funkcijo, katere izhodno vrednost pošljemo naprej po izhodnih povezavah. Nevroni imajo tudi pristranskost, s katero uravnavajo izhodno vrednost neodvisno od vhodnih. Takšne, enostavne mreže so sicer zmožne klasifikacije slik, vendar so za to bolj primerne konvolucijske nevronske mreže, ki so prilagojene posebej za slikovne vhode.

Konvolucijske nevronske mreže so svoj preboj doživele leta 2012 [11], ko so postale glavno orodje za reševanje problema klasifikacije slik. Kot izdaja ime, se konvolucijske mreže zanašajo na konvolucijo, ki poskrbi za ekstrakcijo značilnic iz slike. Zaradi svoje zasnove, konvolucijski filter ohrani prostorske relacije iz vhodne slike. Dodane imajo še zbirne plasti (ang. pooling layers),

ki manjšajo dimenzije značilnic, a ob tem ohranjajo pomembne informacije. Običajno imajo na koncu polno povezane plasti, ki so pravzaprav običajne nevronske mreže, ki smo jih že opisali. Te se učijo nelinearnih razmerij med značilnicami.

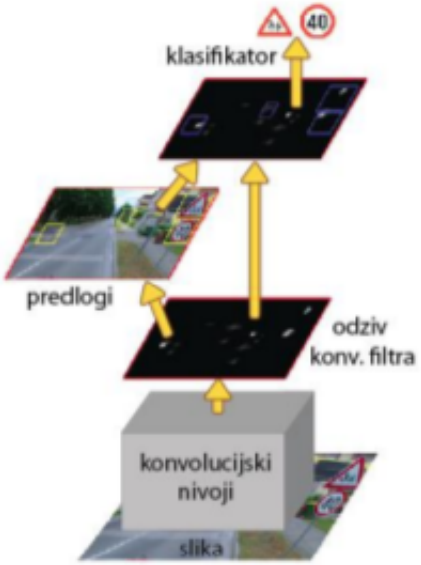
Naš problem je problem detekcije, kjer ne želimo samo prepoznati objekta na sliki, temveč ga tudi locirati. Takšne probleme običajno rešujemo z dvostopenjskim modelom detekcije, ki ga uporablja tudi Faster R-CNN. Namesto na celotno sliko se mreža osredotoči samo na določena območja, regije, kjer se objekti verjetno nahajajo. Slika 2.1 prikazuje takšen dvostopenjski model.

Slika 2.1: Shema dvostopenjske detekcije

Z iskanjem objektov na regijah se uspešnost detekcije močno poveča, vendar se je pojavil nov problem. Kako dobiti ta območja? Nekaj časa smo ta problem reševali z zunanjim generiranjem regij [22], vendar je bilo to časovno zamudno. Mreža Faster R-CNN [17] obdrži dvostopenjski model detekcije, uvede generiranje regij s konvolucijsko nevronsko mrežo (ang. region proposal network, RPN) in poenostavi samo učenje - RPN in klasifikator Fast R-CNN [7] si delita konvolucijske nivoje. Samo generiranje regij sicer upočasni celotno delovanje mreže, vendar je celotno učenje vseeno mnogo hitreje, kot z zunanjim pridobivanjem regij. Nekateri novejši pristopi [16, 13] sicer zmorejo hitrejšo klasifikacijo, a gre tu za pridobivanje hitrosti na račun uspešnosti lokalizacije ali uspešnosti zaznavanja objektov različnih velikosti. Menjava, ki je nismo bili pripravljeni sprejeti.

Če povzamemo: mreža Faster R-CNN je sestavljena iz konvolucijskih nivojev, vrh katerih imamo ločeno generiranje regij in klasifikator. Slednji dobiva vhodne podatke iz konvolucijskih nivojev in RPN. V nadaljevanju

bomo dele podrobno predstavili, nato pa še postopek učenja celotne mreže.

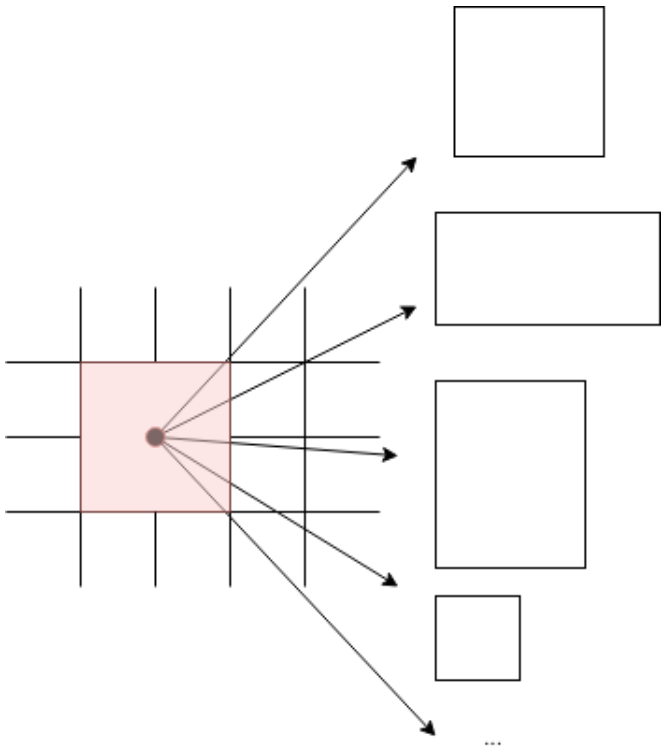


Slika 2.2: Shema Faster R-CNN [15]

2.2 Detekcija regij

Regija je pravokotno območje na sliki, za katero predlagalnik regij, naj si bo to mreža ali kaj drugega, meni da lahko vsebuje objekt, ki ga želimo zaznati. RPN je kot del Faster R-CNN manjša mreža, ki s tehniko drsečega okna (ang. sliding window) sprejme značilnice, ki jih vrne zadnja plast deljenih konvolucijskih nivojev. Detekcijo vršita dve polno povezani plasti. Ti vračata meje regije ter število, ki pove, kako prepričana je mreža, da je na regiji res objekt (ang. objectness score). Do tega pridemo s klasifikacijo objektov v dva razreda - prisoten ali neprisoten objekt. Regije se generirajo na vsakem koraku drsečega okna. Tvorimo 9 sidrišč - to so pravokotniki, s 3 možnimi velikostmi in 3 razmerji med stranicami. Ponazoritev vidimo na 2.3.

Meje objektov določamo glede na ta sidrišča in tako ni potrebno tvoriti piramide slik ali piramide filtrov, hkrati pa s tem dobimo invarianto na



Slika 2.3: Sidrišča, ki jih tvorimo na enem koraku drsečega okna. Zaradi jasnosti so prikazani samo 4, sicer jih je 9.

translacijo.

Na veliki večini realnih slik bodo iskani objekti zasedali le manjši del celote, torej bo iskani objekt na samo nekaj izmed predlaganih regij. Predlagane regije, ki se dovolj prekrivajo z znanimi regijami z znaki (ang. ground truth), uvrščamo v ospredje slike, sicer v ozadje. To razvrstitev uporabimo za izboljšavo učenja: RPN proizvede nekaj tistoč predlaganih regij, učenje in posodabljanje uteži na vseh bi bilo predrago, naključno izbiranje pa bi močno favoriziralo regije iz ozadja. Zato izmed vseh izberemo 256 predlogov za učenje (ang. mini-batch), po vnaprej določenem razmerju regij iz ospredja in ozadja in si tako zagotovimo vidnost regij z objekti. Samo prekrivanje med referenčnimi podatki in predlogi računamo z Jaccardovim indeksom, bolj znanim kot presek nad unijo (ang. Jaccard index, intersection over union).

Za učenje mreže uporabimo kriterijsko funkcijo (ang. loss function), ki je utežena vsota kriterijskih funkcij za klasifikacijo in regresijo (2.1).

$$\begin{aligned} L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) \\ +\lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \end{aligned} \tag{2.1}$$

$$L_{cls}(p, p_i^*) = -\log p_{p_i^*} \tag{2.2}$$

$$L_{reg}(t_i, t_i^*) = \sum_{i \in x,y,w,h} smooth_{L_1}(t_i - t_i^*) \tag{2.3}$$

$$smooth_{L_1} = \begin{cases} 0.5x^2, & \text{če} \mid x \mid < 1 \\ \mid x \mid -0.5, & \text{sicer} \end{cases} \tag{2.4}$$

Na posameznem sidrišču se tvori ocena razreda in parametrizirane koordinate, kjer naj bi se ta nahajal. Sidrišča šteje indeks i, oceno prisotnosti obravnava kriterijska funkcija L_{cls} (2.2), kjer je p_i ocena prisotnosti, ki jo poda mreža. Parameter p_i^* je referenčna vrednost prisotnosti objekta, tako zaseda vrednosti 0 ali 1. Napake v ocenah koordinat kaznuje funkcija L_{reg} [7] (2.3), množenje s p_i^* pa omogoči upoštevanje oz. neupoštevanje ocene glede na dejansko prisotnost objekta. Funkcija L_{reg} se sprehodi po vseh 4 parametrih, ki določajo meje očrtanega pravokotnika x,y,w,h, jim odšteje referenčne vrednosti parametrov in nanje aplicira $smooth_{L1}$. Posamezni izgubi normaliziramo s številom predlogov za učenje (N_{cls}) in številom sidrišč (N_{reg}). Teh je 256 in približno 2400, zato nastavimo λ na 10 in s tem izgubi uravnotežimo.

RPN tako na eni sliki poišče 256 regij, nato pa na vsaki izmed regij in njenih 2400 sidrišč oceni prisotnost objekta in njegove koordinate. Sicer lahko nekaj sidrišč izločimo, ker bodo lezla čez robove slik, vendar jih je vseeno potrebno dodatno filtrirati z odstranjevanjem lokalnih nemaksimumov (ang. non-maximum supression) po oceni razreda. Odstranjevanje lokalnih

nemaksimumov pomeni, da obdržimo med podatki, ki so si blizu samo tisti podatek, ki dosega najvišjo vrednost. Sama definicija bližine je odvisna od aplikacije. Ta določa metriko, po kateri bomo objekt označili za bližnji. Primeri metrik: L1, L2 ali evklidska razdalja, v našem primeru pa Jaccardov indeks. Po odstranjevanju lokalnih nemaksimumov dobimo 2000 očrtanih pravokotnikov (ang. bounding box), ki jih posredujemo naprej v klasifikacijo.

2.3 Klasifikacija objektov

Za klasifikacijo skrbi Fast R-CNN. Konvolucijske plasti so deljene z RPN, nato imamo nekaj zbirnih (ang. pooling) plasti, ki tvorijo vhode za funkcijo softmax in za regresijo očrtanih pravokotnikov, ki bo še nekoliko izboljšala lokalizacijo iskanih objektov. Funkcija softmax (2.5) je posplošena različica logistične funkcije, ki poskrbi, da so vse vrednosti vektorja dožine *k+1*, kjer je k število razredov, v dodan razred pa klasificiramo ozadja. Vrednosti v v so med 0 in 1 in se hkrati seštejejo v 1.

$$\sigma(v_j) = \frac{e^{v_j}}{\sum_{i=1}^{k+1} e^{v_i}} \tag{2.5}$$

Regresija očrtanih pravokotnikov ocenjuje 4k parametrov, 4 koordinate za vsak možen razred. Tudi za Fast R-CNN moramo vzorčiti iz množice predlogov majhno množico učnih primerov, potrebujemo jih 128. Tokrat jih kategoriziramo v ozadje in ospredje glede na drugačne parametre. Če je Jaccardov indeks večji kot 0.5, je pravokotnik del ospredja, med 0.1 in 0.5 pa del ozadja. Regije, ki se sploh ne prekrivajo z referenčnimi podatki zavržemo. Predlagani pravokotniki, ki se prekrivajo z referenčnimi pravokotniki, bodo vsebovali zelo majhen delec znaka. V takšnih primerih se Fast R-CNN večkrat zmoti, zato parameter 0.1 skrbi, da se takšni primeri pojavijo med samim učenjem. Nanj gledamo kot na hevristiko za izbiro težkih primerov. Hkrati predvidevamo, da se bo mreža z učenjem na teh težjih primerih z mnogo ozadja naučila klasificirati tudi regije, kjer je samo ozadje. Tiste, ki smo jih med učenjem zavrgli. Za dodatno izboljšanje učenja v polovici

primerov predlagan pravokotnik zrcalimo horizontalno.

Za učenje mreže uporabimo kriterijsko funkcijo (2.6), ki je utežena vsota kriterijskih funkcij za klasifikacijo in regresijo.

$$L(p, p^*) = L_{cls}(p, p^*) + \lambda[p^* > 0]L_{reg}(t, t^*) \tag{2.6}$$

Funkcija je precej podobna kriterijski funkciji za RPN, razlikuje se le v preverjanju pogoja, ali je referenčni razred na regiji p^* večji kot 0. Razred 0 je namenjen klasifikaciji ozadja. Kriterijska funkcija za klasifikacijo, L_{cls} je enaka kot kriterijska funkcija 2.2, kriterijska funkcija za parametre očrtanih pravokotnikov L_{reg} pa je enaka funkciji 2.3

2.4 Učenje mreže

Faster R-CNN zaradi svoje zgradbe terja nov pristop učenja. Konvolucijske plasti si morajo deliti tako RPN kot Fast R-CNN. Želimo si, da bi te plasti učili za klasifikacijo vseh razredov, torej za Fast R-CNN, a pred tem moramo detektirati regije z RPN. Zato mrežo učimo v večih korakih. Prvi korak mora biti učenje za RPN. Mrežo inicializiramo z utežmi za klasifikacijo podatkovne zbirke ImageNet [5], kot je prikazano v [7]. Po tem koraku imamo konvolucijske plasti naučene za klasifikacijo regij v ozadja in ospredja in RPN del mreže, ki je proizvedla regije za nadaljnjo klasifikacijo. V drugem koraku zavržemo naučene uteži in jih ponovno inicializiramo z utežmi iz ImageNeta. Sedaj učimo te plasti in Fast R-CNN, z regijami ki smo jih pridobili v prvi fazi. Od te točke konvolucijske plasti zamrznemo, jih ne učimo več. Tretji korak moramo izvesti, ker je RPN uglašen na plasti, ki smo jih zavrgli, torej ga je potrebno ponovno učiti z izhodi novih konvolucijskih plasti. Sedaj imamo naučene konvolucijske plasti in RPN. V četrtem koraku učimo Fast R-CNN z regijami iz RPN, ki je sedaj uglašen s konvolucijskimi plasti. Tretji in četrti korak bi sicer lahko ponavljali še naprej v upanju, da se bodo rezultati izboljševali, a eksperimenti kažejo, da ne pride do večjega izboljšanja.

Štiri koraki učenja na kratko:

1. Konvolucijske plasti inicializiramo z ImageNet utežmi, učimo samo ta del in RPN mrežo.

2. Inicializiramo nove konvolucijske plasti z ImageNet utežmi, uporabimo predlagane regije iz prvega koraka, učimo Fast R-CNN del mreže.

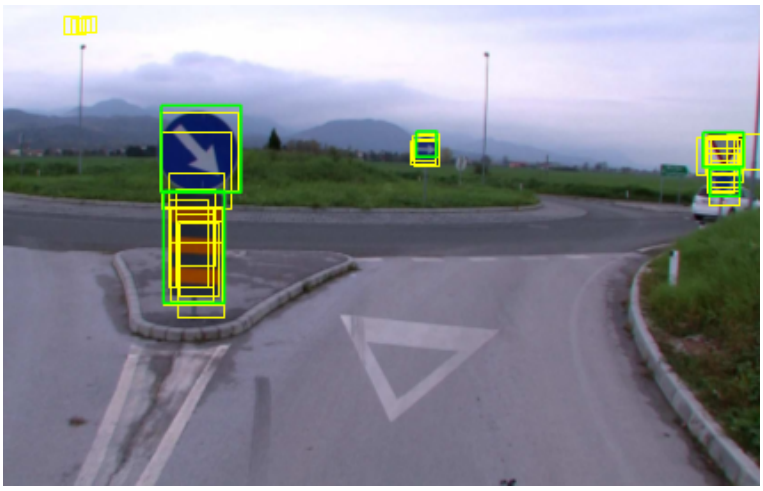
3. Konvolucijske plasti iz prvega koraka nadomestimo s tistimi iz drugega koraka, vendar zamrznemo njene uteži. Ker je mreža zamenjana, je potrebno doučiti RPN mrežo, da si spet prilegata, oziroma da se RPN izmojstri (ang. fine tuning) v predlaganju regij glede na nove plasti.

4. V zadnjem koraku mojstrimo še klasifikator Fast R-CNN.

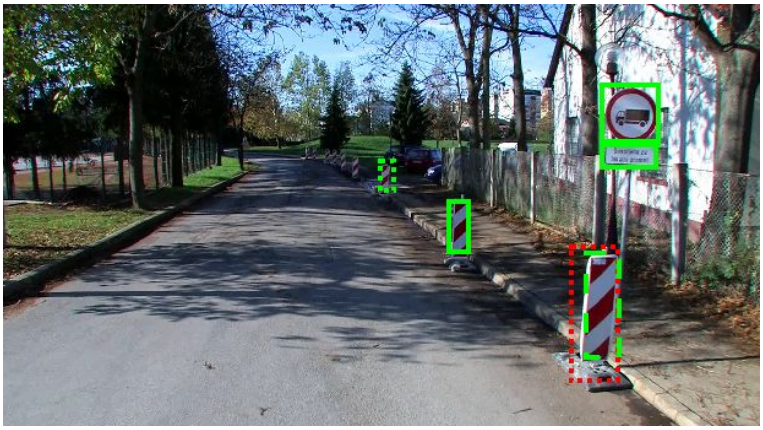
2.5 Uporaba mreže

Ob predpostavki, da imamo model že naučen, moramo za klasificiranje znakov na sliki samo poskrbeti, da je ta prave velikosti. Podatke lahko vizualiziramo, kot je vidno na sliki 2.4. Prikazali smo predlagane regije, vidne na sliki kot pravokotniki. Na sliki je 30 regij, za katere je mreža najbolj prepričana, da vsebujejo znake. Za potrditev svoje odločitve uporabi pragovne vrednosti za razrede, ki misli da so na sliki. Če presežejo pragovne vrednosti mreža trdi, da se v tej regiji nahaja znak. Na sliki so te regije označene z zeleno barvo.

Slika 2.5 prikazuje slabše detektirano sliko. Z rdečimi obrobami vidimo napačno zaznavo, tako klasificirano zaradi neprilegajočega se očrtanega pravokotnika.



Slika 2.4: Prikaz detektiranja slike. [15]



Slika 2.5: Slika s pravimi in lažnimi detekcijami.

3. Razširitve

Za učenje nevronskih mrež potrebujemo veliko podatkov. Te nato delimo v tri množice - učno, glede na katero se mreža uči, validacijsko, kjer določamo hiperparametre, in testno, kjer ugotavljamo, kako uspešno je bilo učenje. Zelo pomembno je, da so si množice tuje, saj želimo, da se mreža nauči klasificirati nove primerke, torej ne sme biti preveč prilagojena na našo učno množico (ang. overfitting). V prvem delu se bomo ukvarjali z validacijskimi množicami, v drugem delu z večanjem učne množice skozi čas, v tretjem delu bomo raziskali možnost izboljšanja učenja s sprotnim iskanjem težkih primerov.

3.1 Določanje optimalnih pragovnih vrednosti

Naša mreža, oziroma klasifikator na koncu, nam vrača oceno verjetnosti, da je na sliki določen prometni znak. Ker si želimo enostaven odgovor, znak je prisoten ali pa ni, moramo določiti pragovne vrednosti, glede na katere bomo ta odgovor oblikovali. Če bo ocenjena verjetnost višja kot pragovna vrednost, bomo trdili, da znak na sliki je, sicer ga naj ne bi bilo. Pragovne vrednosti navadno določamo na validacijski množici, saj bi se na učni množici preveč prilagodili učnim primerom, testna množica pa se uporablja samo za testiranje. Tako bomo na validacijski množici poiskali optimalne pragovne vrednosti za različne metrike in hkrati raziskali možnost uporabe umetno generiranih slik.

Uspešnost klasifikacije lahko merimo na različne načine, glede na željene lastnosti klasifikatorja, oziroma namen aplikacije. Za glavno merilo smo uporabili f-mero, ki upošteva tako priklic kot natančnost, nato smo preverili priklic in natančnost še vsako posebej.

Natančnost pove, koliko izmed vseh zaznanih pozitivnih primerov je resnično pozitivnih.

$$\text{natančnost} = \frac{\text{resnični pozitivni}}{\text{resnični pozitivni } + \text{ lažni pozitivni}} \tag{3.1}$$

Priklic nam pove, kolikšen delež vseh pozitivnih primerov je naš sistem zaznal.

$$\text{priklic} = \frac{\text{resnični pozitivni}}{\text{resnični pozitivni } + \text{lažni negativni}} \tag{3.2}$$

F-mera je harmonično povprečje priklica in natančnosti, pomnoženo z 2. S tem dosežemo, da je maksimalna vrednost f-mere 1.

$$\begin{aligned} F &= 2 * \frac{1}{\frac{1}{priklic} + \frac{1}{\text{natančnost}}} \\ &= 2 * \frac{\text{priklic } * \text{ natančnost}}{\text{priklic} + \text{natančnost}} \end{aligned} \tag{3.3}$$

Tvorili bomo validacijske množice iz resničnih in umetno generiranih slik, različnih velikosti. Na njih bomo računali pragovne vrednosti, ki jih bomo preizkusili na testni množici. Dobre pragovne vrednosti prinesejo dobro klasifikacijo testnih slik in upravičujejo uporabo neke validacijske množice.

3.2 Doučenje z novimi podatki

Včasih se zgodi, da nekaj časa po uspešnem učenju in vpeljavi sistema v rabo dobimo nove podatke, ki bi morda lahko izboljšali naš model. Zanimalo nas je, kako bi jih uporabili, da bi dobili najboljši rezultat - v našem primeru, maksimizirali f-mero. Nove podatke bi lahko združili s starimi in ponovno naučili model, ali pa bi vzeli obstoječ model in ga uporabili za učenje z novimi podatki, ga doučili. V ta namen smo učne podatke razdelili na dva dela. Najprej smo naučili model na polovici podatkov, nato smo ga doučili

z drugo polovico. V primeru Faster R-CNN doučenje pomeni, da v drugem koraku učenja inicializiramo konvolucijske plasti z utežmi iz učenja na prejšnjih podatkih. Tvorili smo dve učni množici, ena je predstavljala prvo učenje, druga pa pozneje pridobljene slike. Preizkušali smo različno dolga učenja na drugem delu in kombiniranje prve in druge množice za doučenje.

3.3 Sprotno iskanje težkih primerov

Ena izmed izboljšav učenja je namensko učenje mreže na težkih primerih [14] (ang. bootstrapping, hard example mining). Navadno to pomeni alterniranje med učenjem mreže na učni množici in iskanjem novih težkih primerov, s katerimi jo bomo dopolnili. Med iskanjem teh težkih primerov, slik, ki jih naša mreža narobe klasificira, moramo ustaviti učenje in testirati stotine slik, da naberemo dovolj slabo klasificiranih. Te potem dodamo v učno množico, iz nje odstranimo najbolje klasificirane slike in proces večkrat ponavljamo. Dandanes so nevronske mreže sestavljene iz mnogo plasti, tudi Faster R-CNN, zato učenje navadno terja več deset ali sto tisoč iteracij. Tako vsaka ustavitev učenja povzroči zamudo. Zato se je pojavila potreba po sprotnem iskanju težkih primerov, po nečem, kar ne bi terjalo dolgotrajnega zamrzovanja modela in prečesavanja slik.

Metoda sprotnega iskanja težkih primerov [19] (Online Hard Example Mining, OHEM) temelji na predpostavki, da je v vsaki učni množici veliko večino primerov lahko kategorizirati, le manjši del se izkaže za težkega. Prej smo povedali, da dobi klasifikator 2000 regij, ki jih mora klasificirati. V našem primeru smo zato izbirali četrtino regij iz ospredja in tri četrtine iz ozadja. To pomeni, da smo morali ročno nastaviti tri parametre - dva pogoja za klasifikacijo slik v ozadje in ospredje ter enega, ki nam določi razmerja v končnem izboru regij. Vsak ročno nastavljen parameter učenja pomeni neko hevristiko, ki bo za naš primer morda delovala, ali pa tudi ne. Metoda OHEM opusti ročno nastavljanje filtriranja regij in uvede izbiranje regij glede na to, kako slabo jih klasifikator klasificira. Sam postopek je sila enostaven: vseh

2000 regij spustimo skozi mrežo (ang. forward pass), s tem dobimo klasifikacijsko točnost in posledično izgubo za vsako regijo. Nato regije razporedimo padajoče po izgubi in odstranimo lokalne nemaksimume, kjer je maksimum maksimalna izguba, bližino pa določa Jaccardov indeks. Nato vzamemo 128 najslabše klasificiranih regij in na njih učimo mrežo. Za OHEM je odstranjevanje lokalnih nemaksimumov pomembno, saj izgube pri prekrivajočih se regijah korelirajo, zaradi česar bi ista napaka zasedala več mest med 128 regijami za učenje. Naš mini-batch je tako enake velikosti kot običajno, kar pomeni, da časovno potratno posodabljanje uteži ohrani isto ceno. OHEM bi lahko vpeljali tako na RPN kot na klasifikator Fast R-CNN, vendar smo se zaradi cene računanja izgub na vseh regijah (0.5s) odločili za implementacijo, kot jo navaja izvoren članek, samo na klasifikatorju.

Psevdokoda sprotnega iskanja težkih primerov:

begin

$net.outputs = net.forwardPass(vse\_regije)$;

izračun predikcij za razrede

$izgube = izracunajIzgube(net.outputs)$;

izračun izgub glede na predikcije

$izgube = sort(izgube)$;

uredi izgube padajoče, največje najprej

$mini\_batch = non\_max\_suppresion(izgube)$;

odstranjevanje regij, ki korelirajo

$mini\_batch = mini\_batch[1:128]$;

omejimo se na 128 najslabših regij

$net.step(mini\_batch)$;

učenje mreže na filtrirani podmnožici vseh regij

4. Eksperimentalni rezultati

4.1 Eksperimentalni podatki

4.1.1 Zbirka slik

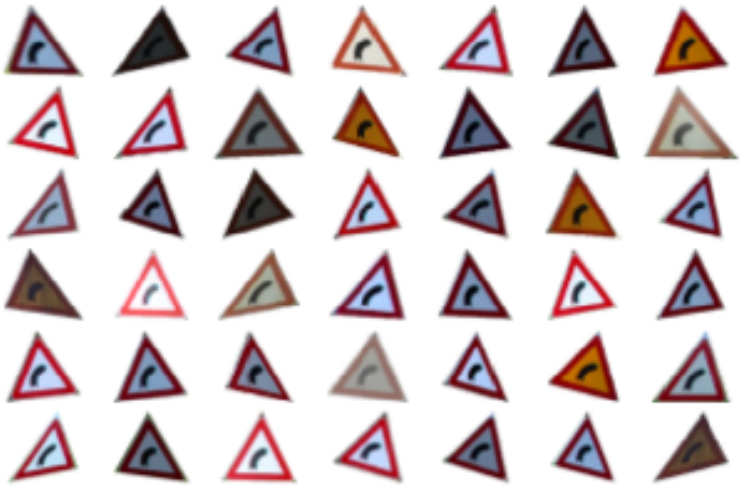
Za učno množico smo uporabili 6165 slik prometnih znakov, posnetih na slovenskih cestah. Zajeli smo jih med vožnjo skozi 6 občin. Izmed vseh slik smo izbrali samo tiste, ki vsebujejo znake. Pozorni smo bili, da je med pari slik večja menjava ozadja, da smo se izognili sekvenci slik približevanja prometnemu znaku. Slike, ki so bile posnete manj kot 50m narazen smo dali v iste gruče. Te slike si bi lahko bile preveč podobne, zato v izogib podvojevanju slik v različnih množicah gruč ob deljenju nismo razbijali. Delili smo jih v tri množice - testno (1244 slik), validacijsko (1418 slik) in učno (2857 slik). Za učenje smo vzeli 160 katogorij znakov, 123 jih ima standardno obliko, vedno enaka razmerja stranic, določeno vsebino. 37 kategorij je težjih, to so razne krajevne table, zrcala in podobno, kjer je vsebina precej variabilna. Vsi učni primeri so v očrtanem pravokotniku, katerega koordinate se poskušamo naučiti. Najkrajša stranica tega pravokotnika je vedno velika vsaj 30 slikovnih elementov. Ker torej od mreže ne terjamo, da se nauči znakov, manjših od te dolžine, smo vse manjše zaznave ob evalvaciji filtrirali. Tako so vsi rezultati prikazani za zaznave, ki imajo stranice dolge vsaj 30 slikovnih elementov, razen kjer opozorimo na drugačno določino stranic.

Učenja smo izvajali na treh grafičnih karticah Nvidia Titan X, tako smo hkrati lahko izvajali 3 eksperimente. Polno učenje je trajalo 3 dni za en poskus, ena validacija pa 6 ur.

Slika 4.1: Nabor znakov naše množice. Zgoraj levo so prikazane lahke kategorije, pod njimi težje. Zgoraj desno primer učne slike, desno na sredini in spodaj prikaz znakov v realnih pogojih.

4.1.2 Umetno generiranje podatkov za validacijo

Slike smo generirali iz znakov, ki smo jih izrezali iz realnih slik. Na izrezanih znakih smo uporabili različne transformacije, kot je vidno na sliki 4.2. Spreminjali smo osvetlitev, barve, kontraste, uporabili smo tudi afine preslikave. Za ozadja smo uporabili slike iz tuje baze [21], ki niso vsebovale znakov. Slike iz te baze so bile posnete v Belgiji, tako da se precej razlikujejo od naših slik, posnetih v Sloveniji. Generirane predloge in ozadja smo zlepili skupaj in tako dobili nove učne primere. Generirane slike so precej bolj tuje učni množici kot testne slike, izhajajo iz drugega okolja, posnete so bile z drugo opremo, imajo slabšo ločljivost.



Slika 4.2: Transformirani znaki, izrezani iz resničnih slik. [15]

4.2 Določanje optimalnih pragovnih vrednosti

Da bi preverili vpliv uporabe generiranih znakov smo uporabili prej opisano delitev, učno množico naučili, nato pa isto mrežo validirali na različnih validacijskih množicah in nato vedno na isti testni. Validacijske množice imajo dva različna vira, znake smo luščili iz validacijske in učne množice. Znaki, izluščeni iz validacijske množice bodo popolnoma tuji učni množici, tisti iz učne množice pa se bodo razlikovali po ozadju in apliciranih transformacijah. Ker lahko generiramo poljubno veliko umetnih slik, smo poskusili podvojiti velikost validacijske množice in preverili, kako in če sploh ta spremeba vpliva na rezultate. Dodali smo še dva primera, ki preverjata vpliv velikosti učne množice na rezultat, da vidimo, ali je sploh smiselno iskati načine večanja množice.

Učili in validirali smo na množicah, prikazani v tabeli 4.1:

oznaka	učna množica	validacijska množica	vir validacijske množice
1	1880	1650	izvorna validacijska množica
2	2850	1650	izvorna validacijska množica
3	2850	1650	igenerirano iz validacijske množice
4	2850	1650	generiranih učne množice
5	2850	3300	generiranih iz validacijske množice
6	4730	3300	generiranih iz učne množice

Tabela 4.1: Množice, na katerih smo določali optimalne pragovne vrednosti

4.2.1 Maksimizacija f-mere

Ob analizi rezultatov smo ugotovili, da ima 5% zaznanih znakov najkrajšo stranico krajšo od 30 slikovnih elementov, kar 44% pa krajšo kot 50 slikovnih elementov. Zato smo pri izračunu f-mere preverili rezultate na obeh množicah. V tabeli 4.2 smo v prvem stolpcu upoštevali vse zaznave, kjer je najkrajša stranica dolga vsaj 30 slikovnih elementov, v drugem stolpcu pa imajo vsi zaznani znaki vsaj 50 slikovnih elementov na najkrajši stranici. Za našo aplikacijo lahko zaznave pod 50 elementov zanemarimo, saj je zelo verjetno, da bodo to narobe klasificirani znaki, deli ozadja. Če smo vseeno zaznali znak, manjši kot 50 slikovnih elementov, ga bomo zaznali pozneje, ko mu bomo nekoliko bližje. Zaznane znake manjše kot 30 slikovnih elementov lahko odstranimo, saj v učni množici sploh nimamo tako majhnih znakov, pod to mejo je večina zaznav lažnih.

Primera 1 in 2 prikazujeta vpliv velikosti učne množice na f-mero. Vidimo, da večja množica res vpliva na rezultat. Primera 3 in 4 imata validacijsko množico enake velikosti kot prejšnja primera, podatki, ki so generirani iz učne množice dajejo nekoliko slabši rezultat kot tisti iz validacijske množice. Primera 5 in 6 imata prav tako generirano validacijsko množico, vendar dvakratne velikosti prvotne. Vidimo, da je v vseh primerih umetno generirana validacijska množica približno enako uspešna ali boljša kot iz resničnih slik.

učenje	>30	>50
1	0.79	0.80
2	0.83	0.84
3	0.84	0.87
4	0.84	0.87
5	0.85	0.88
6	0.85	0.87
7	0.87	0.90

Tabela 4.2: F-mere glede na različne validacijske množice in velikosti učnih množic ter velikost zaznanih znakov

Vidimo, da je izboljšanje konsistentno skozi vse umetno generirane množice. V primeru 2 v primerjavi s 3 in 4 razliko pripisujemo tujosti ozadij validacijske množice. Za dodaten test smo vzeli 7. primer, kjer uporabimo izvorno validacijsko množico kot del učne, za validacijo pa uporabimo umetno generiranje podatkov. Tudi tu se f-mera zviša, kar opravičuje uporabo generiranih slik za validacijsko množico v prid večanju učne množice. Filtriranje v tem primeru pokaže, da lahko glede na potrebe aplikacije rezultate poljubno zvišamo.

4.2.2 Maksimalna natančnost s pripadajočim priklicem

Maksimizacijo natančnosti uporabljamo v aplikacijah, kjer želimo biti čim bolj prepričani, da je zaznan znak resnično znak. Ob tem si seveda želimo kar se da največji priklic, torej zaznavanje znakov, vendar raje vidimo, da kakšnega izpustimo, kot da jih prepoznavamo prepogosto.

Maksimum natančnosti dosežemo že v drugem primeru, kjer uporabljamo celotno učno množico in realne validacijske slike. V nadaljnjih primerih generirana validacijska množica dviga maksimalno natančnost. Sicer je v sedmem primeru priklic malo manjši kot v primerih 2-7, vendar je to verjetno zaradi naključnih inicializacij uteži in naključnega razdeljevanja slik. Zdi se, da je v

učenje	natančnost	priklic
1	0.96	0.63
2	0.99	0.68
3	0.99	0.75
4	0.99	0.76
5	0.99	0.73
6	0.99	0.75
7	0.98	0.79

Tabela 4.3: Maksimalna natančnost s pripadajočim priklicem

tem primeru generiranje slik iz učne množice boljše, saj vidimo da je primer 4 boljši od 3, ravno tako 6 boljši od 5. Razlog, da je 5. primer slabši od ostalih primerov, kjer umetno generiramo slike, pripisujemo naključnosti samega generiranja. Potrebno bi bilo narediti več preizkusov in rezultate povprečiti, da bi določili točen vzrok. Če vzamemo podatke kot so, se moramo odločiti za četrti primer, kjer za narobe označimo kot znak le 1%, ob tem pa izgubimo 24% znakov, ki so bili tam, a jih zavoljo šibkosti našega prepričanja zavržemo.

4.2.3 Maksimalen priklic s pripadajočo natančnostjo

Za našo aplikacijo je bolj kot maksimizacija natančnosti pomembna maksimizacija priklica. Želimo si izpustiti čim manj znakov, ob tem pa ni tako pomembno, če kak drug objekt označimo kot znak. Napačne zaznave lahko pozneje odstranimo s kakšno drugo obliko verifikacije.

Rezultati v tabeli 4.4 kažejo, da na izboljšanje priklica vpiva samo velikost učne množice. Rezultati od 2 do 7 imajo enake vrednosti in enako učno množico. Prvi primer ima nekoliko manjši priklic in pripadajočo natančnost, učna množica za ta primer je 20% manjša. Izboljšanje v primerjavi z drugim, osnovnim primerom, dosežemo šele v 7. primeru, kjer smo povečali učno

učenje	natančnost	priklic
1	0.75	0.92
2	0.79	0.94
3	0.79	0.94
4	0.79	0.94
5	0.79	0.94
6	0.79	0.94
7	0.86	0.96

Tabela 4.4: Maksimalen priklic s pripadajočo natančnostjo.

množico. Če v naši aplikaciji uporabimo 7. primer, bomo izpustili 4% znakov, 14% pa bomo morali ročno filtrirati iz zaznanih znakov.

4.3 Doučenje z novimi podatki

Tudi v tem poskusu smo uporabili delitev slik, opisano v prvem delu tega poglavja. Učno množico smo zatem delili na dva približno enaka dela, s čimer smo oponašali poznejšo pridobitev teh podatkov. Ob tem smo se soočali s problemi večih znakov na isti sliki. To nam je onemogočalo lepo enakomerno delitev, zato smo jo izvedli požrešno: vse znake smo prešteli in tako dobili koliko znakov naj bi imeli v posamezni kategoriji, v idealnem primeru. Nato smo slike dodelili drugi množici, če je kateri izmed znakov na njej že dosegel maksimalno število v prvi množici, sicer smo jih dodelili v prvo množico. V nadaljevanju bomo prvo polovico označevali s T1, drugo s T2. Zaradi narave delitve je v množici T1 tako eden izmed razredov popolnoma brez primerkov, 42 razredov ima manj kot 10 učnih primerov. V množici T2 je minimum 7 primerkov, manj kot 10 primerov imajo 4 razredi. Ker je bila osnovna ideja, da zajamemo množico T2 pozneje kot T1, se nam je zdelo bolje, da so v T2 bolje zastopani redki primeri. Na primeru naše aplikacije, kjer poskušamo detektirati znake, je verjetneje, da bomo skozi čas naleteli

na redkejše primerke, kot da jih dobimo že prvič v učni množici, nato pa v vožnji po vseh cestah naletimo samo še na pogostejše.

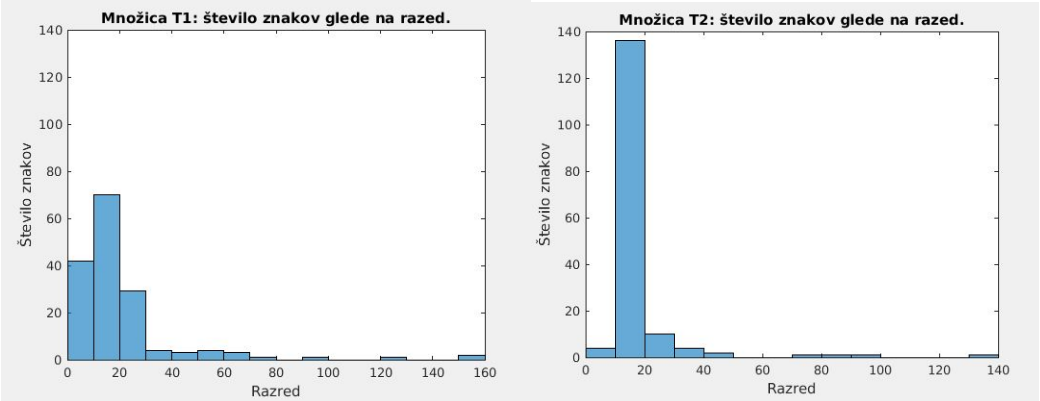
Iz množic T1 in T2 smo tvorili več kombinacij za doučenje, vidnih v tabeli 4.5. Primer 0 je osnovno učenje na polovici podatkov. V primeru 1 doučimo T1 z množico T2, v 2. primeru doučimo s kombinacijo prve in druge polovice. Pri prvem primeru obstaja možnost, da se bo celotna mreža preveč prilagodila na množico T2, saj pri doučenju več ne bi videla množice T1. Druga množica je sestavljena iz množice T1 in T2, vendar je zaradi prejšnjega učenja morda bolje naučena za detekcije primerov iz T1. 3. množica poskuša odpraviti razliko med številom videvanj slik iz T1 in T2, s podvojevanjem še nevidenih primerov iz T2.

oznaka	začetno učenje	doučeno z množico
0	T1	/
1	T1	T2
2	T1	T1 in T2
3	T1	T1,T2,T2
4	T1,T2	/

Tabela 4.5: Množice, s katerimi smo preverjali vpliv doučenja.

Množice za doučenje niso unije, v tretjem primeru bo prišlo do podvajanja elementov iz T2. Četrti primer je kontrolni, uporabimo navadno učenje z običajnim številom iteracij, s celotno množico slik. Za nadaljnjo razlago moramo razložiti pojem iteracija. Iteracija je eno učenje mreže, kjer za vhodne podatke uporabimo regije iz mini-batcha. Običajno število iteracij - 80 000 za RPN in 40 000 za Fast R-CNN smo za osnovno učenje in doučenje prepolovili in tako dobili 40 000 in 20 000 iteracij. Tako ostane število iteracij na posamezno sliko približno enako pri doučenju in v navadnem primeru.

Učenja smo evalvirali petkrat, glede na število učnih primerov na razred. Najprej brez filtiranja po razredih, torej na 160 razredov. Z zahtevo, da ima vsak razred več kot 10 znakov za vsak razred, nam ostane za evalvacijo 89



(a) Število učnih primerov v osnovi za učenje,T1.

(b) Število učnih primerov v množici za poznejše učenje, T2.

Slika 4.3: Delitev učnih primerov v množici T1 in T2.

razredov. Z zahtevo minimalnih 15 učnih primerov v obeh množicah dobimo 33 razredov, če jih ima vsak več kot 20 dobimo 18 razredov, nad 25 primerov jih ima 12. Analizirali smo maksimalno f-mero vsakega od učenj, pri analizi smo opazovali predvsem zvišanje/zmanjšanje te glede na osnovno učenje 0. Primerjava konkretnih f-mer med učenji z različnimi števili razredov je težavnejša, zaradi različne narave znakov in različnih težavnosti klasifikacije različnih razredov.

		število	znakov	v	razredu
	vse	>10	>15	>20	>25
0	0.86	0.83	0.81	0.80	0.82
1.	0.80	0.83	0.83	0.85	0.85
2.	0.84	0.84	0.85	0.85	0.87
3.	0.82	0.84	0.85	0.87	0.88
4.	0.83	0.84	0.84	0.84	0.83

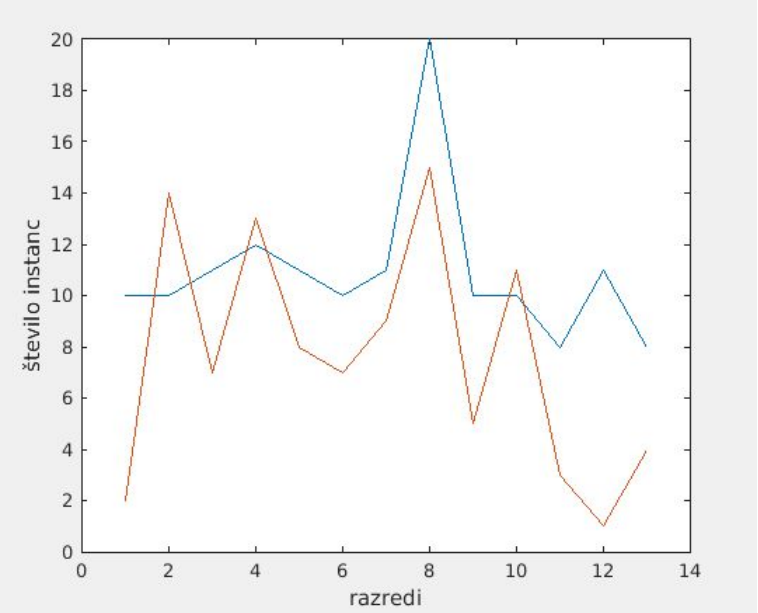
Tabela 4.6: F-mera glede na uporabljeno množico za doučenje, glede na minimalno število učnih primerov na razred.

Vidimo, da so rezultati v prvem stolpcu zelo nenavadni. Na samo polovici podatkov dosežemo f-mero 0.86, z nadaljnjimi doučenji pa se ta zniža. Preverili smo, kateri znaki so v primeru 0 bolje kategorizirani po doučenju, izstopalo je 13 znakov. Ti so imeli f-mero večjo za vsaj 20% v primeru 0, kot primerih 1-4. Na sliki 4.4 so prikazani ti znaki.

Slika 4.4: Znaki, ki ki imajo v T1 f-mero višjo za več kot 0.2 v primerjavi z ostalimi primeri.

Preverili smo, kako so ti znaki številčno zastopani v primerih 0 in 1. Graf 4.5 prikazuje razliko v številu učnih primerov med 0, prvo polovico za učenje in 1, s katero smo mrežo doučili pozneje. Vidimo, da je skoraj v vseh primerih število instanc manjše v 0. Ker je naša mreža pri učenju na 0 videla manj primerov teh znakov, je ob klasifikaciji objektov v te razrede bolj konzervativna. To se kaže v manjšem številu lažnih zaznav teh znakov v primeru 0. Število lažno zaznanih objektov, torej zaznavanje znakov, kjer jih v resnici ni, se močno dvigne v vseh primerih doučenja oziroma v vseh primerih, tudi v 4. primeru. Na grafu vidimo tudi, da je večina teh znakov v vsakem setu 15 ali manj, tako da bo že filtriranje po tej vrednosti zmanjšalo njihov vpliv. Filtriranje, kjer zahtevamo vsaj 20 instanc v obeh polovicah bo

njihov učinek popolnoma izničilo. Predvidevamo, da bi ob močnem zvišanju števila teh znakov število lažnih zaznav padlo.



Slika 4.5: Rumena črta predstavlja število učnih primerov v T1, modra število instanc v T2.

V vseh ostalih primerih prinese doučenje v primerjavi s ponovnim učenjem na celotnih podatkih izboljšanje v f-meri. T1 je naša osnovna uspešnost, z doučenjem te želimo doseči rezultat celotnega učenja, 4. primer, ali ga izboljšati. V prvem primeru doučimo z množico tujo začetni množici T1, s čimer poslabšamo ali kvečjemu izenačimo rezultat, ki ga doseže ponovno učenje celotne množice. Drugi primer uporabi za doučenje množico T1 in T2, rezultati so boljši od ponovnega učenja v primeru, ko se omejimo na razrede z več kot 25 primerki. V tretjem primeru sestavlja množico za doučenje T1 in podvojena množica T2. Ta je v vseh primerih boljša od ponovnega učenja celotne množice.

4.4 Sprotno iskanje težkih primerov

Sprotno iskanje težkih primerov (v nadaljevanju OHEM) smo preizkusili na dva načina. Najprej tako kot je algoritem mišljen, z izbiranjem najslabše klasificiranih regij, ne glede na to, ali so v ozadju ali ospredju. Ker se ta pristop ni najbolje obnesel, smo poskusili samo z izbiranjem najslabših regij iz ozadja. Regije iz ospredja smo prepustili navadnemu izbiranju, saj ponavadi število regij iz ospredja ne preseže odmerjene četrtine mini-batcha, torej 32 regij. Zanimalo nas je tudi, ali z uporabo pristopa OHEM naš model hitreje konvergira, zato smo v nekaterih primerih prepolovili število iteracij. Preverili smo tudi spremembe v f-meri glede na minimalno velikost zaznanih znakov. Pragove smo, tako kot pri iskanju optimalnih pragovnih vrednosti, postavili na 30 in 50 slikovnih elementih. Na sliki 4.6 smo prikazali primer napačne zaznave, ki jo lahko, s primerno omejitvijo velikost zaznanih objektov, uspešno odstranimo.

Slika 4.6: Slika prikazuje uspešno zaznavo, očrtano z zeleno in napačno, v ozadju s črtkano rdečo črto. Velikost napačne zaznave je 45 slikovnih elementov po najkrajši stranici.

Rezultati kažejo, da uporaba OHEMa na vseh regijah poslabša rezultat, kot je vidno ob primerjavi prve vrstice s tretjo, ter četrte s šesto. Izbira-

brez/ozadje/vse regije	pol/vse iteracije	>30	>50
brez	pol	0.80	0.82
ozadje	pol	0.80	0.82
vse regije	pol	0.76	0.81
brez	vse	0.82	0.84
ozadje	vse	0.84	0.85
vse regije	vse	0.80	0.78

Tabela 4.7: F-mera glede na uporabo sprotnega iskanja težkih primerov in število iteracij, za zaznane znake, ki imajo najkrajšo stranico večjo od 30 slikovnih elementov.

nje regij za učenje glede na neuspešnost klasifikacije se obrestuje samo, če z metodo sprotnega iskanja težkih primerov izbiramo primere na ozadju. Ena izmed prednosti te metode je zmanjševanje hiperparametrov, torej parametrov, ki v tem primeru določajo koliko regij iz ospredja in ozadja vzamemo in kaj sploh štejemo kot ospredje in ozadje. Kadar uporabimo to okrnjeno verzijo, ki vzorči samo iz ozadja, to izboljšavo zavržemo, saj moramo te hiperparametre ohraniti.

Polnokrvni OHEM, ki izbira vse regije glede na največjo izgubo, v našem primeru ne deluje tako dobro, saj bo skoraj vedno izbral manj regij iz ospredja kot običajno vzorčenje. Da je klasifikacija slabša ravno zaradi teh, vidimo po razlikah med izbiranjem iz vseh regij in samo tistimi iz ozadja. Razlog je v tem, da bo OHEM oklestil število regij z odstranjevanjem lokalnih nemaksimumov in bo tako imel na voljo manjše število regij z znaki kot brez tega. Hkrati pri običajnem vzorčenju vedno vzamemo za učenje vsaj četrtino regij iz ospredja, kar je za večino slik celo več, kot jih vsebuje, ali pa vsaj večina. OHEM tako ne more izbrati za učenje več regij iz ospredja kot navadno vzorčenje.

5. Zaključek

V delu smo detektirali prometne znake z metodo Faster R-CNN in iskali možne razširitve. Preizkušali smo različne validacijske množice. Ugotovili smo, da jo lahko napolnimo z umetno ustvarjenimi slikami in si tako resnične slike prihranimo za učno množico. Preverili smo možnost doučenja z novimi podatki in našli optimalno zgradbo za to množico. Ob tem smo naleteli na nenavadno višanje f-mere na manjši učni množici. Nazadnje smo preizkusili delovanje metode sprotnega iskanja težkih primerov. Zdi se, da metoda za naš problem ni primerna, saj celo zniža rezultate klasifikacije.

Zaradi relativno velikega števila učenj in omejitev strojne opreme smo vsako učenje izvedli le enkrat. Resnejša statistična obdelava podatkov bi terjala več ponovitev vsakega izmed eksperimentov.

Literatura

[1] Holger Caesar, Jasper R. R. Uijlings, and Vittorio Ferrari. Region-based semantic segmentation with end-to-end training. *CoRR*, abs/1607.07671, 2016.

[2] Zhaowei Cai, Quanfu Fan, Rogério Schmidt Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. *CoRR*, abs/1607.07155, 2016.

[3] Neelima Chavali, Harsh Agrawal, Aroma Mahendru, and Dhruv Batra. Object-proposal evaluation protocol is 'gameable'. *CoRR*, abs/1505.05836, 2015.

[4] Dan C. Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. A committee of neural networks for traffic sign classification. In *IJCNN*, pages 1918–1921. IEEE, 2011.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. IEEE Computer Society, 2009.

[6] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[7] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[8] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[9] Jan Hendrik Hosang, Rodrigo Benenson, Piotr Dollár, and Bernt Schiele. What makes for effective detection proposals? *CoRR*, abs/1502.05082, 2015.

[10] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. How good are detection proposals, really? *CoRR*, abs/1406.6962, 2014.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[12] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[14] Constantine Papageorgiou and Tomaso Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 38(1):15–33, Jun 2000.

[15] Rok Mandeljc Danijel Skočaj Peter Uršič, Domen Tabernik. Detekcija velikega Števila kategorij prometnih znakov. In *Zbornik strokovne konference ROSUS 2017*, pages 59–67, 2017.

[16] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[17] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[18] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[19] Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training region-based object detectors with online hard example mining. *CoRR*, abs/1604.03540, 2016.

[20] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323 – 332, 2012. Selected Papers from IJCNN 2011.

[21] Radu Timofte, Karel Zimmermann, and Luc van Gool. Multi-view traffic sign detection, recognition, and 3d localisation. In *Ninth IEEE Computer Society Workshop on Application of Computer Vision*, pages 1–8, Snowbird, Utah, USA, December 2009.

[22] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[23] Yingying Zhu, Chengquan Zhang, Duoyou Zhou, Xinggang Wang, Xiang Bai, and Wenyu Liu. Traffic sign detection and recognition using fully convolutional network guided proposals. *Neurocomputing*, 214:758 – 766, 2016.